Megapixel Topology Optimization using *fW*–mean Filters

Linus Hägg and Eddie Wadbro

Department of Computing Science
Umeå University

BIT Circus 2015 Umeå

Material distribution (topology optimization)

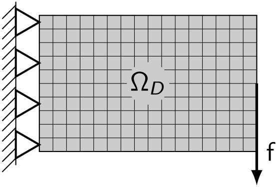
- Distribute material arbitrarily in the design domain Ω_D
- Material distribution function ρ ("density") constant in each element

Example

Linear elasticity, minimization of compliance

- $\rho_i = 0$ if void and 1 if solid
- Want to solve:

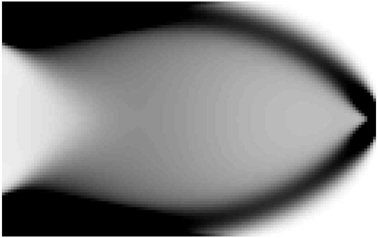
$$\begin{aligned} \min_{\boldsymbol{\rho}} \quad & J(\boldsymbol{\rho}) \quad \text{(compliance)} \\ \text{s.t.} \quad & \rho_i \in \{0,1\} \;\; \forall i \\ & \frac{1}{n}\sum_{i=1}^{n} \rho_i \le V \\ & \text{governing PDE} \end{aligned}$$



Typical approach

Large-scale non-linear integer optimization problems are very hard to solve. . .

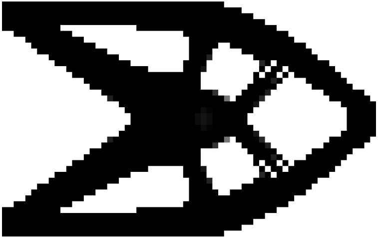
- Relaxation: $0 \leq \rho_i \leq 1$
- Penalization: force ρ_i to either 0 or 1
  - Mesh dependence
- Filtering
  - Minimum length scale
  - Intermediate densities
  - Computationally expensive



Typical approach

Large-scale non-linear integer optimization problems are very hard to solve...

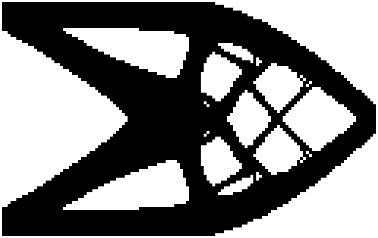
- Relaxation: $0 \leq \rho_i \leq 1$
- Penalization: force ρ_i to either 0 or 1
  - Mesh dependence
- Filtering
  - Minimum length scale
  - Intermediate densities
  - Computationally expensive



Typical approach

Large-scale non-linear integer optimization problems are very hard to solve. . .

- Relaxation: $0 \leq \rho_i \leq 1$
- Penalization: force ρ_i to either 0 or 1
  - Mesh dependence
- Filtering
  - Minimum length scale
  - Intermediate densities
  - Computationally expensive



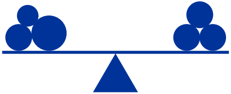
Typical approach

Large-scale non-linear integer optimization problems are very hard to solve...

- Relaxation: $0 \leq \rho_i \leq 1$
- Penalization: force ρ_i to either 0 or 1
  - Mesh dependence
- Filtering
  - Minimum length scale
  - Intermediate densities
  - Computationally expensive

Quasi-arithmetic means (f-means)

- Weighted arithmetic mean
  $M_x(\boldsymbol{x};\boldsymbol{w}) = \boldsymbol{w}^T\boldsymbol{x} \equiv \sum_{i=1}^{n} w_i x_i$
  $\boldsymbol{w}^T\mathbf{1}_{\boldsymbol{n}} \equiv \sum_{i=1}^{n} w_i = 1$
  $w_i > 0$
- Weighted harmonic mean
  $M_{x^{-1}}(\boldsymbol{x};\boldsymbol{w}) = \left(\boldsymbol{w}^T\boldsymbol{x}^{-\mathbf{1}}\right)^{-1}$
- Weighted geometric mean
  $M_{\ln x}(\boldsymbol{x};\boldsymbol{w}) = \prod_{i=1}^{n} x_i^{w_i} \equiv \exp\left(\boldsymbol{w}^T\mathbf{ln}\,\boldsymbol{x}\right)$
- ...



Quasi-arithmetic mean (f-mean)

$$M_f(\boldsymbol{x};\boldsymbol{w}) = f^{-1}\left(\boldsymbol{w}^T\boldsymbol{f}(\boldsymbol{x})\right) \iff f(M_f) = \boldsymbol{w}^T\boldsymbol{f}(\boldsymbol{x})$$

fW-mean filters (Wadbro and Hägg, 2015)

Replace the value of the design variable in one element with the f-mean of the values of its neighboring elements:

fW-mean filter

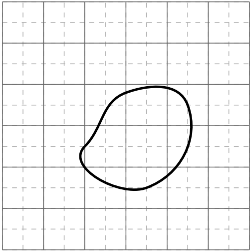
- $\boldsymbol{F}(\rho) = \boldsymbol{f}^{-1}(\boldsymbol{W}\boldsymbol{f}(\rho))$
  $\boldsymbol{W} = [w_{ij}] \in \mathbb{R}^{n \times n}$
  $w_{ij} \geq 0$ and $\boldsymbol{W}\mathbf{1_n} = \mathbf{1_n}$
- $w_{ij} > 0$ iff $j \in \mathcal{N}_i \subset \{1, \ldots, n\}$
- Replace f^{-1} with g, then a vast majority of available filters
  - Heaviside filter (Guest et al., 2004),
  - Morphology-based filters (Sigmund 2007),
  - Pythagorean mean based filters (Svanberg and Svärd, 2014)

  can be handled in a similar manner

Computational complexity of *fW*-mean filtering

- d-dimensional Cartesian grid with n elements of size h
- Neighborhoods with *fixed physical size* $V \propto (hr)^d$
  - r is the filter radius (measured in number of elements)
- Computational complexity: $O(n|\mathcal{N}_i|)$

$$\begin{cases} |\mathcal{N}_i| \propto h^{-d}(hr)^d \\ r^d \propto n \end{cases} \Rightarrow O(n|\mathcal{N}_i|) = O(n^2)$$



To achieve $O(n)$, additional assumptions on $\boldsymbol{W}$ and the geometry of the neighborhoods are needed

Equal weight fW-mean filters

- $\boldsymbol{W} = \boldsymbol{D}^{-1}\boldsymbol{G}$
  $\boldsymbol{D} = \mathrm{diag}(|\mathcal{N}_1|, \ldots, |\mathcal{N}_n|)$
  $\boldsymbol{G} = [g_{ij}]$
  - $g_{ij} = 1$ if $j \in \mathcal{N}_i$
  - $g_{ij} = 0$ if $j \notin \mathcal{N}_i$
- Split: $\boldsymbol{F}(\rho) = \boldsymbol{f}^{-1}\left(\boldsymbol{D}^{-1}\boldsymbol{G}\boldsymbol{f}(\rho)\right)$

  (1) $\boldsymbol{a} = \boldsymbol{f}(\rho)$

  (2) $\boldsymbol{s} = \boldsymbol{G}\boldsymbol{a} \iff s_i = \sum_{k \in \mathcal{N}_i} a_k$

  (3) $\boldsymbol{F}(\rho) = \boldsymbol{f}^{-1}\left(\boldsymbol{D}^{-1}\boldsymbol{s}\right)$

Note that (1) and (3) can be performed element-wise, (2) only involves summation. Hence (1) and (3) are $O(n)$. How about the "neighborhood sums" in (2)?

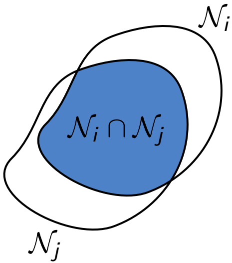
Basic idea for fast evalution of neighborhood sums

Q: Given $\sum_{k\in\mathcal{N}_i}$ how should we compute $\sum_{k\in\mathcal{N}_j}$?

- Compute from scratch
  Complexity: $|\mathcal{N}_j|$
- Update using:

$$\sum_{k\in\mathcal{N}_j} = \sum_{k\in\mathcal{N}_i} + \sum_{k\in\mathcal{N}_j\setminus\mathcal{N}_i} - \sum_{k\in\mathcal{N}_i\setminus\mathcal{N}_j}$$

Complexity: $|\mathcal{N}_j \setminus \mathcal{N}_i| + |\mathcal{N}_i \setminus \mathcal{N}_j| = |\mathcal{N}_j| + |\mathcal{N}_i| - 2|\mathcal{N}_i \cap \mathcal{N}_j|$

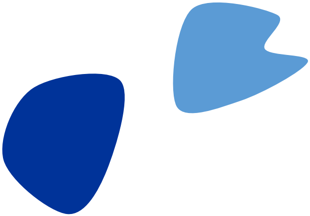


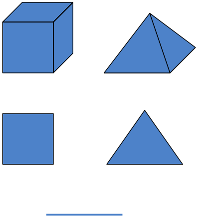
Conclusion: If the overlap is large updating is favorable.
The next question is: How favorable?

The significance of neighborhood geometry

Q: Given $\sum_{k \in \mathcal{N}_i}$, how many operations are needed to compute the sum at an *adjacent* element using the update strategy?

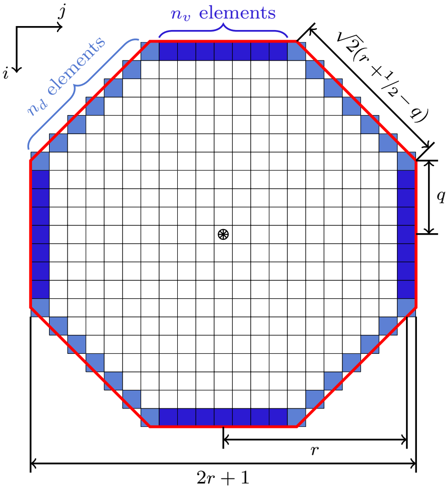
- Only elements on the *boundaries* of the two neighborhoods are needed ($O(r^{d-1})$ elements)
  - Computational complexity: $O(n^{2-1/d})$
- The facets of a polytope in $\mathbb{R}^d$ are polytopes in $\mathbb{R}^{d-1}$
  - Perform updates in a recursive manner!





Fast summation over octagonal neighborhoods

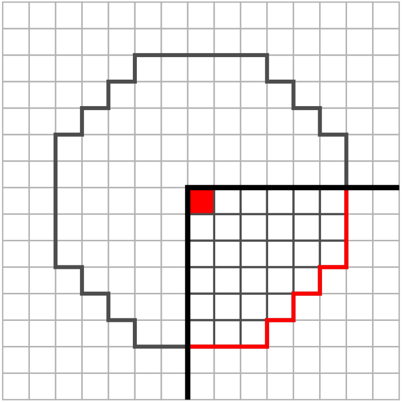
- $n = n_1 n_2$
- $r < \min\{n_1/2, n_2/2\}$
- $a_{i,j}$ summands
- $s_{i,j}$ sums



Fast summation over octagonal neighborhoods

Step 1

- Compute $s_{1,1}$ from scratch
- Complexity: $< (r+1)^2$



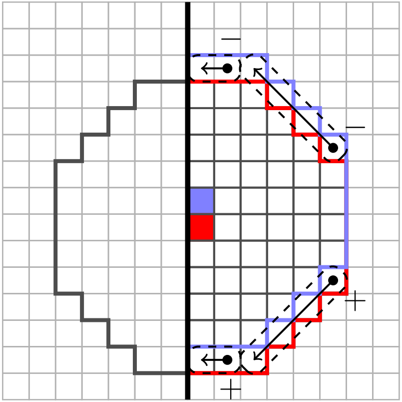
Fast summation over octagonal neighborhoods

Step 1

- Compute $s_{1,1}$ from scratch
- Complexity: $< (r+1)^2$

Step 2

- Compute $s_{i,1}$ by updating $s_{i-1,1}$
- Complexity: $4(n_1 - 1)$



Fast summation over octagonal neighborhoods

Step 1

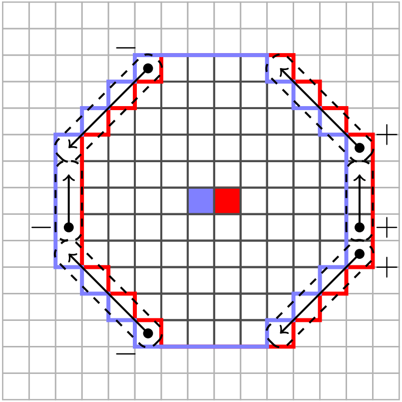
- Compute $s_{1,1}$ from scratch
- Complexity: $< (r+1)^2$

Step 2

- Compute $s_{i,1}$ by updating $s_{i-1,1}$
- Complexity: $4(n_1 - 1)$

Step 3

- Compute $s_{i,j}$ by updating $s_{i,j-1}$
- Complexity: $6n_1(n_2 - 1)$



Fast summation over octagonal neighborhoods

Step 1

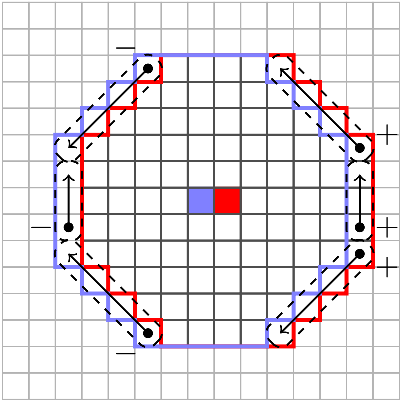
- Compute $s_{1,1}$ from scratch
- Complexity: $< (r+1)^2$

Step 2

- Compute $s_{i,1}$ by updating $s_{i-1,1}$
- Complexity: $4(n_1 - 1)$

Step 3

- Compute $s_{i,j}$ by updating $s_{i,j-1}$
- Complexity: $6n_1(n_2 - 1)$



Complexity of each 1D-sum: $< 2n$

Fast summation over octagonal neighborhoods

Step 1

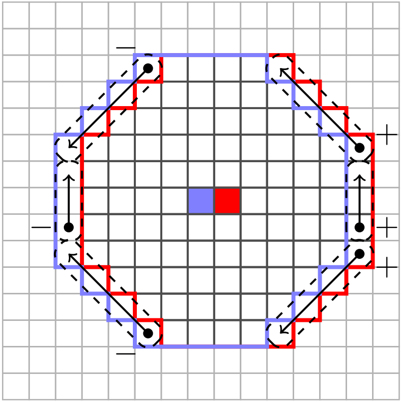
- Compute $s_{1,1}$ from scratch
- Complexity: $< (r+1)^2$

Step 2

- Compute $s_{i,1}$ by updating $s_{i-1,1}$
- Complexity: $4(n_1 - 1)$

Step 3

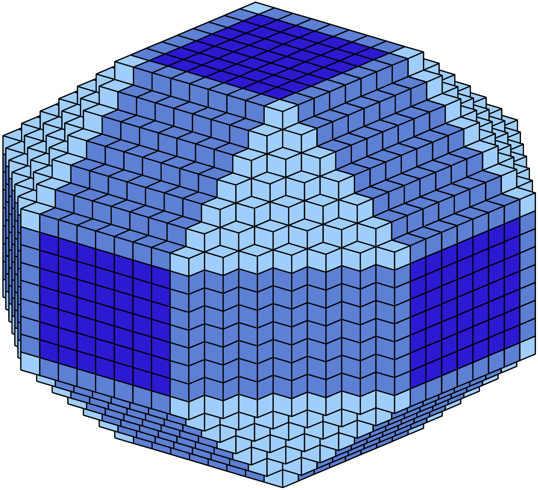
- Compute $s_{i,j}$ by updating $s_{i,j-1}$
- Complexity: $6n_1(n_2 - 1)$



Complexity of each 1D-sum: $< 2n$

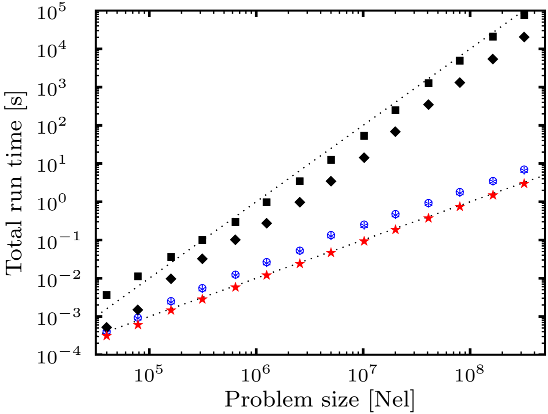
Overall complexity (to leading order): $< 13n$

Rhombicuboctahedral neighborhoods



Overall complexity (to leading order): $< 61n$

Execution times for various problem sizes

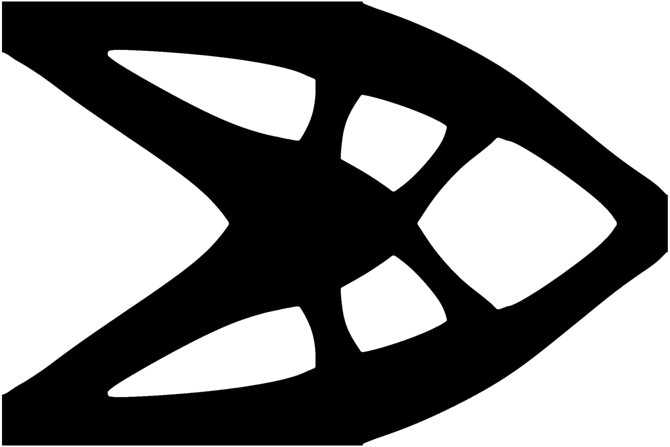


Standard $O(n^2)$ algorithm

Proposed $O(n)$ algorithm

Baseline: create and write n random numbers to memory

Harmonic open-close megapixel optimization



- $2160 \times 1440 \approx 3.11 \cdot 10^6$ design variables
- Four consequtive *fW*–mean filters provide independent size control on both material and void regions